Clustering on Chocolate Bar Ratings

Table of Contents

#### Author: Andrew Tran

## Blog Post Inspiration and Objectives

In this blog post, I am exploring basic methods of clustering in Machine Learning. One interesting topic that I came across included a dataset thousands of different types of chocolate bars found across the world globally, including all other documented factors such as the Company that made it, the origin of the cocoa bean used in it, the year of the chocolate bar review, the percentage of cocoa computed in the composition of the bar, etc. I thought that it would be interesting to see if I can determine if I can figure out any corrleation relationships between the features of the chocolate bar and its numerical rating through means of clustering. My logic is that similar-featured chocolate bars would be grouped in the same cluster. However, this may lead to concerns in determining which factors (if any) weigh more in terms of swaying the rating of that particular chocolate bar entry. Thus, I soon began my blog post, trying to find some way to resolve this question. With that said, let’s try to analyze this topic with some Machine Learning:

## Data Preprocessing - Cleaning and Analytics

```{python}  
# Import needed libraries  
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
%matplotlib inline  
from matplotlib.collections import LineCollection  
import seaborn as sns  
color = sns.color\_palette()  
from sklearn.preprocessing import MinMaxScaler, StandardScaler  
from sklearn.pipeline import Pipeline  
from sklearn.model\_selection import train\_test\_split  
from yellowbrick.cluster import KElbowVisualizer  
from sklearn.cluster import MiniBatchKMeans, KMeans, DBSCAN  
from sklearn.neighbors import NearestNeighbors  
import kneed  
from sklearn.decomposition import PCA  
from sklearn.metrics import average\_precision\_score, roc\_auc\_score, precision\_recall\_fscore\_support, confusion\_matrix, classification\_report  
from sklearn.metrics import mean\_squared\_error, pairwise\_distances  
plt.style.use("fivethirtyeight")  
```

First, we will read and display the initial dataset in our file system for this blog post, downloaded from Kaggle. This dataset contains loads of valuable information such as all notable chocolate-bar quantiative statistics that you would typically document such as Company that created the bar being reviewed, REF number, Year of Review Date, Cocoa Percent in bar, Company Location of the bar being reviewed, Bean Type, and Cocoa Bean origin of the company that distributed the bar being reviewed.

```{python}  
# Reading and displaying the initial dataset (ignoring any warnings or errors)  
df = pd.read\_csv("datasets/flavors\_of\_cacao.csv")  
df  
```

|  | Company \n(Maker-if known) | Specific Bean Origin\nor Bar Name | REF | Review\nDate | Cocoa\nPercent | Company\nLocation | Rating | Bean\nType | Broad Bean\nOrigin |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | A. Morin | Agua Grande | 1876 | 2016 | 63% | France | 3.75 |  | Sao Tome |
| 1 | A. Morin | Kpime | 1676 | 2015 | 70% | France | 2.75 |  | Togo |
| 2 | A. Morin | Atsane | 1676 | 2015 | 70% | France | 3.00 |  | Togo |
| 3 | A. Morin | Akata | 1680 | 2015 | 70% | France | 3.50 |  | Togo |
| 4 | A. Morin | Quilla | 1704 | 2015 | 70% | France | 3.50 |  | Peru |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 1790 | Zotter | Peru | 647 | 2011 | 70% | Austria | 3.75 |  | Peru |
| 1791 | Zotter | Congo | 749 | 2011 | 65% | Austria | 3.00 | Forastero | Congo |
| 1792 | Zotter | Kerala State | 749 | 2011 | 65% | Austria | 3.50 | Forastero | India |
| 1793 | Zotter | Kerala State | 781 | 2011 | 62% | Austria | 3.25 |  | India |
| 1794 | Zotter | Brazil, Mitzi Blue | 486 | 2010 | 65% | Austria | 3.00 |  | Brazil |

For clarity on the constraints and parameters of the working datasets, I went to find high-level exploratory statistics on all of the datasets: shape, information about all of the entries, etc.

```{python}  
# Determining the shape of the initial dataset  
df.shape  
```

(1795, 9)

```{python}  
# Getting a sample of the initial dataset through the seeing the first 10 entries  
# completely in the dataset  
df.head()  
```

|  | Company \n(Maker-if known) | Specific Bean Origin\nor Bar Name | REF | Review\nDate | Cocoa\nPercent | Company\nLocation | Rating | Bean\nType | Broad Bean\nOrigin |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | A. Morin | Agua Grande | 1876 | 2016 | 63% | France | 3.75 |  | Sao Tome |
| 1 | A. Morin | Kpime | 1676 | 2015 | 70% | France | 2.75 |  | Togo |
| 2 | A. Morin | Atsane | 1676 | 2015 | 70% | France | 3.00 |  | Togo |
| 3 | A. Morin | Akata | 1680 | 2015 | 70% | France | 3.50 |  | Togo |
| 4 | A. Morin | Quilla | 1704 | 2015 | 70% | France | 3.50 |  | Peru |

```{python}  
# Figuring out all of the columns (and their names) available for me to use in   
# the dataset  
df.columns  
```

Index(['Company \n(Maker-if known)', 'Specific Bean Origin\nor Bar Name',  
 'REF', 'Review\nDate', 'Cocoa\nPercent', 'Company\nLocation', 'Rating',  
 'Bean\nType', 'Broad Bean\nOrigin'],  
 dtype='object')

```{python}  
# Getting basic information about the dataset  
df.info()  
```

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 1795 entries, 0 to 1794  
Data columns (total 9 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 Company   
(Maker-if known) 1795 non-null object   
 1 Specific Bean Origin  
or Bar Name 1795 non-null object   
 2 REF 1795 non-null int64   
 3 Review  
Date 1795 non-null int64   
 4 Cocoa  
Percent 1795 non-null object   
 5 Company  
Location 1795 non-null object   
 6 Rating 1795 non-null float64  
 7 Bean  
Type 1794 non-null object   
 8 Broad Bean  
Origin 1794 non-null object   
dtypes: float64(1), int64(2), object(6)  
memory usage: 126.3+ KB

```{python}  
# Figuring out the number of duplicated elements in the dataset (could be   
# problematic if not resolved)  
df.duplicated().sum()  
```

0

Additionally, before handing my Financial-Institution Fraud dataset over for Machine Learning training and prediction, I need to clean the data prior to the analysis stage: removing duplicates, deleting null/NaN values, fixing types of columns, filling invalid values with suitable alternatives, etc.

```{python}  
# Renaming the columns to be more readable   
cols\_rename\_dict = {}  
for col in df.columns:  
 cols\_rename\_dict.update({col: col.replace("\n", " ")})  
df = df.rename(columns=cols\_rename\_dict)  
  
df.rename(columns={"REF": "Reference Number"}, inplace=True)  
  
cols\_rename\_dict = {}  
for col in df.columns:  
 cols\_rename\_dict.update({col: col.replace(" (Maker-if known)", "")})  
df = df.rename(columns=cols\_rename\_dict)  
df.columns = df.columns.str.strip()  
  
df  
```

|  | Company | Specific Bean Origin or Bar Name | Reference Number | Review Date | Cocoa Percent | Company Location | Rating | Bean Type | Broad Bean Origin |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | A. Morin | Agua Grande | 1876 | 2016 | 63% | France | 3.75 |  | Sao Tome |
| 1 | A. Morin | Kpime | 1676 | 2015 | 70% | France | 2.75 |  | Togo |
| 2 | A. Morin | Atsane | 1676 | 2015 | 70% | France | 3.00 |  | Togo |
| 3 | A. Morin | Akata | 1680 | 2015 | 70% | France | 3.50 |  | Togo |
| 4 | A. Morin | Quilla | 1704 | 2015 | 70% | France | 3.50 |  | Peru |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 1790 | Zotter | Peru | 647 | 2011 | 70% | Austria | 3.75 |  | Peru |
| 1791 | Zotter | Congo | 749 | 2011 | 65% | Austria | 3.00 | Forastero | Congo |
| 1792 | Zotter | Kerala State | 749 | 2011 | 65% | Austria | 3.50 | Forastero | India |
| 1793 | Zotter | Kerala State | 781 | 2011 | 62% | Austria | 3.25 |  | India |
| 1794 | Zotter | Brazil, Mitzi Blue | 486 | 2010 | 65% | Austria | 3.00 |  | Brazil |

```{python}  
# Figuring out the number of 'null'/'NaN' elements in the dataset (i.e. if NaN   
# filling is needed or not)  
print(df.isnull().sum())  
(df.isnull().sum() / df.shape[0]) \* 100  
```

Company 0  
Specific Bean Origin or Bar Name 0  
Reference Number 0  
Review Date 0  
Cocoa Percent 0  
Company Location 0  
Rating 0  
Bean Type 1  
Broad Bean Origin 1  
dtype: int64

Company 0.00000  
Specific Bean Origin or Bar Name 0.00000  
Reference Number 0.00000  
Review Date 0.00000  
Cocoa Percent 0.00000  
Company Location 0.00000  
Rating 0.00000  
Bean Type 0.05571  
Broad Bean Origin 0.05571  
dtype: float64

```{python}  
for col in df.columns:  
 display(df[col].unique())  
```

array(['A. Morin', 'Acalli', 'Adi', 'Aequare (Gianduja)', 'Ah Cacao',  
 "Akesson's (Pralus)", 'Alain Ducasse', 'Alexandre',  
 'Altus aka Cao Artisan', 'Amano', 'Amatller (Simon Coll)',  
 'Amazona', 'Ambrosia', 'Amedei', 'AMMA', 'Anahata', 'Animas',  
 'Ara', 'Arete', 'Artisan du Chocolat',  
 'Artisan du Chocolat (Casa Luker)', 'Askinosie', 'Bahen & Co.',  
 'Bakau', 'Bar Au Chocolat', "Baravelli's", 'Batch', 'Beau Cacao',  
 'Beehive', 'Belcolade', 'Bellflower', 'Belyzium', 'Benoit Nihant',  
 'Bernachon', 'Beschle (Felchlin)', 'Bisou', 'Bittersweet Origins',  
 'Black Mountain', 'Black River (A. Morin)', 'Blanxart',  
 'Blue Bandana', 'Bonnat', 'Bouga Cacao (Tulicorp)', 'Bowler Man',  
 "Brasstown aka It's Chocolate", 'Brazen', 'Breeze Mill', 'Bright',  
 'Britarev', 'Bronx Grrl Chocolate', 'Burnt Fork Bend',  
 'Cacao Arabuco', 'Cacao Atlanta', 'Cacao Barry', 'Cacao de Origen',  
 'Cacao de Origin', 'Cacao Hunters', 'Cacao Market', 'Cacao Prieto',  
 'Cacao Sampaka', 'Cacao Store',  
 'Cacaosuyo (Theobroma Inversiones)', 'Cacaoyere (Ecuatoriana)',  
 'Callebaut', 'C-Amaro', 'Cao', 'Caoni (Tulicorp)',  
 'Captain Pembleton', 'Caribeans', 'Carlotta Chocolat',  
 'Castronovo', 'Cello', 'Cemoi', 'Chaleur B', 'Charm School',  
 'Chchukululu (Tulicorp)', 'Chequessett', 'Chloe Chocolat',  
 'Chocablog', 'Choco Del Sol', 'Choco Dong', 'Chocolarder',  
 "Chocola'te", 'Chocolate Alchemist-Philly', 'Chocolate Con Amor',  
 'Chocolate Conspiracy', 'Chocolate Makers', 'Chocolate Tree, The',  
 'Chocolats Privilege', 'ChocoReko', 'Chocosol', 'Chocovic',  
 'Chocovivo', 'Choklat', 'Chokolat Elot (Girard)', 'Choocsol',  
 'Christopher Morel (Felchlin)', 'Chuao Chocolatier',  
 'Chuao Chocolatier (Pralus)', 'Claudio Corallo', 'Cloudforest',  
 'Coleman & Davis', 'Compania de Chocolate (Salgado)', 'Condor',  
 'Confluence', 'Coppeneur', "Cote d' Or (Kraft)", 'Cravve', 'Creo',  
 'Daintree', 'Dalloway', 'Damson', 'Dandelion', 'Danta', 'DAR',  
 'Dark Forest', 'Davis', 'De Mendes', 'De Villiers',  
 'Dean and Deluca (Belcolade)',  
 'Debauve & Gallais (Michel Cluizel)', 'Desbarres', 'DeVries',  
 'Dick Taylor', 'Doble & Bignall', 'Dole (Guittard)',  
 'Dolfin (Belcolade)', 'Domori', 'Dormouse', "Duffy's", 'Dulcinea',  
 'Durand', 'Durci', 'East Van Roasters', 'Eau de Rose',  
 'Eclat (Felchlin)', 'Edelmond', 'El Ceibo', 'El Rey',  
 'Emerald Estate', "Emily's", 'ENNA',  
 'Enric Rovira (Claudio Corallo)', 'Erithaj (A. Morin)', 'Escazu',  
 "Ethel's Artisan (Mars)", 'Ethereal', 'Fearless (AMMA)',  
 'Feitoria Cacao', 'Felchlin', 'Finca', 'Forever Cacao',  
 'Forteza (Cortes)', 'Fossa', 'Franceschi', 'Frederic Blondeel',  
 'French Broad', 'Fresco', 'Friis Holm', 'Friis Holm (Bonnat)',  
 'Fruition', 'Garden Island', 'Georgia Ramon', 'Glennmade',  
 'Goodnow Farms', 'Grand Place', "Green & Black's (ICAM)",  
 'Green Bean to Bar', 'Grenada Chocolate Co.', 'Guido Castagna',  
 'Guittard', 'Habitual', 'Hachez', 'Hacienda El Castillo', 'Haigh',  
 'Harper Macaw', 'Heilemann',  
 'Heirloom Cacao Preservation (Brasstown)',  
 'Heirloom Cacao Preservation (Fruition)',  
 'Heirloom Cacao Preservation (Guittard)',  
 'Heirloom Cacao Preservation (Manoa)',  
 'Heirloom Cacao Preservation (Millcreek)',  
 'Heirloom Cacao Preservation (Mindo)',  
 'Heirloom Cacao Preservation (Zokoko)', 'hello cocoa', 'hexx',  
 'Hogarth', 'Hoja Verde (Tulicorp)', 'Holy Cacao', 'Honest',  
 'Hotel Chocolat', 'Hotel Chocolat (Coppeneur)', 'Hummingbird',  
 'Idilio (Felchlin)', 'Indah', 'Indaphoria', 'Indi', 'iQ Chocolate',  
 'Isidro', 'Izard', 'Jacque Torres', 'Jordis',  
 'Just Good Chocolate', 'Kah Kow', 'Kakao', 'Kallari (Ecuatoriana)',  
 'Kaoka (Cemoi)', 'Kerchner', "Ki' Xocolatl", 'Kiskadee', 'Kto',  
 "K'ul", 'Kyya', 'L.A. Burdick (Felchlin)',  
 'La Chocolaterie Nanairo', 'La Maison du Chocolat (Valrhona)',  
 'La Oroquidea', 'La Pepa de Oro', 'Laia aka Chat-Noir',  
 'Lajedo do Ouro', 'Lake Champlain (Callebaut)', "L'Amourette",  
 'Letterpress', 'Levy', 'Lilla', 'Lillie Belle', 'Lindt & Sprungli',  
 'Loiza', 'Lonohana', 'Love Bar', 'Luker',  
 'Machu Picchu Trading Co.', 'Madecasse (Cinagra)', 'Madre',  
 'Maglio', 'Majani', 'Malagasy (Chocolaterie Robert)', 'Malagos',  
 'Malie Kai (Guittard)', 'Malmo', 'Mana', 'Manifesto Cacao',  
 'Manoa', 'Manufaktura Czekolady', 'Map Chocolate', 'Marana',  
 "Marigold's Finest", 'Marou', 'Mars', 'Marsatta', 'Martin Mayer',  
 'Mast Brothers', 'Matale', 'Maverick', 'Mayacama', 'Meadowlands',  
 'Menakao (aka Cinagra)', 'Mesocacao', 'Metiisto', 'Metropolitan',  
 'Michel Cluizel', 'Middlebury', 'Millcreek Cacao Roasters',  
 'Mindo', 'Minimal', 'Mission', 'Mita', 'Moho', 'Molucca',  
 'Momotombo', 'Monarque', 'Monsieur Truffe', 'Montecristi',  
 'Muchomas (Mesocacao)', 'Mutari', 'Nahua', 'Naive', 'Na�ve',  
 'Nanea', 'Nathan Miller', 'Neuhaus (Callebaut)', 'Nibble',  
 'Night Owl', 'Noble Bean aka Jerjobo', "Noir d' Ebine",  
 'Nova Monda', 'Nuance', 'Nugali', 'Oakland Chocolate Co.', 'Obolo',  
 'Ocelot', 'Ocho', 'Ohiyo', 'Oialla by Bojessen (Malmo)',  
 'Olive and Sinclair', 'Olivia', 'Omanhene', 'Omnom', 'organicfair',  
 'Original Beans (Felchlin)', 'Original Hawaiin Chocolate Factory',  
 'Orquidea', 'Pacari', 'Palette de Bine', 'Pangea', 'Park 75',  
 'Parliament', 'Pascha', 'Patric', 'Paul Young', 'Peppalo',  
 'Pierre Marcolini', 'Pinellas', 'Pitch Dark',  
 'Pomm (aka Dead Dog)', 'Potomac', 'Pralus', 'Pump Street Bakery',  
 'Pura Delizia', 'Q Chocolate', 'Quetzalli (Wolter)', 'Raaka',  
 'Rain Republic', 'Rancho San Jacinto', 'Ranger', 'Raoul Boulanger',  
 'Raw Cocoa', 'Republica del Cacao (aka Confecta)', 'Ritual',  
 'Roasting Masters', 'Robert (aka Chocolaterie Robert)',  
 'Rococo (Grenada Chocolate Co.)', 'Rogue', 'Rozsavolgyi',  
 'S.A.I.D.', 'Sacred', 'Salgado', 'Santander (Compania Nacional)',  
 'Santome', 'Scharffen Berger', 'Seaforth', 'Shark Mountain',  
 "Shark's", 'Shattel', 'Shattell', 'Sibu', 'Sibu Sura',  
 'Silvio Bessone', 'Sirene', 'Sjolinds', 'Smooth Chocolator, The',  
 'Snake & Butterfly', 'Sol Cacao', 'Solkiki', 'Solomons Gold',  
 'Solstice', 'Soma', 'Somerville', 'Soul', 'Spagnvola', 'Spencer',  
 'Sprungli (Felchlin)', 'SRSLY', 'Starchild',  
 'Stella (aka Bernrain)', 'Stone Grindz', 'StRita Supreme',  
 'Sublime Origins', 'Summerbird', 'Suruca Chocolate',  
 'Svenska Kakaobolaget', 'Szanto Tibor', 'Tabal',  
 'Tablette (aka Vanillabeans)', 'Tan Ban Skrati', 'Taza', 'TCHO',  
 'Tejas', 'Terroir', 'The Barn', 'Theo', 'Theobroma',  
 'Timo A. Meyer', "To'ak (Ecuatoriana)", 'Tobago Estate (Pralus)',  
 'Tocoti', 'Treehouse', 'Tsara (Cinagra)', 'twenty-four blackbirds',  
 'Two Ravens', 'Un Dimanche A Paris', 'Undone', 'Upchurch', 'Urzi',  
 'Valrhona', 'Vanleer (Barry Callebaut)',  
 'Vao Vao (Chocolaterie Robert)', 'Vicuna', 'Videri',  
 'Vietcacao (A. Morin)', 'Vintage Plantations',  
 'Vintage Plantations (Tulicorp)', 'Violet Sky', 'Vivra',  
 'Wellington Chocolate Factory', 'Whittakers', "Wilkie's Organic",  
 "Willie's Cacao", 'Wm', 'Woodblock', 'Xocolat', 'Xocolla', "Zak's",  
 'Zart Pralinen', 'Zokoko', 'Zotter'], dtype=object)

array(['Agua Grande', 'Kpime', 'Atsane', ..., 'Indianer, Raw',  
 'Kerala State', 'Brazil, Mitzi Blue'], dtype=object)

array([1876, 1676, 1680, 1704, 1315, 1319, 1011, 1015, 1019, 797, 1462,  
 1470, 705, 370, 316, 636, 502, 508, 1215, 1061, 1173, 1944,  
 1728, 1732, 1125, 1129, 1133, 725, 470, 544, 363, 304, 129,  
 147, 175, 322, 327, 464, 1145, 1494, 1498, 979, 111, 123,  
 170, 40, 75, 1065, 572, 1259, 1852, 1375, 1379, 1724, 1900,  
 1904, 1908, 1924, 1928, 1534, 1598, 1602, 1193, 947, 729, 745,  
 486, 531, 600, 355, 300, 1181, 1780, 647, 661, 331, 141,  
 1474, 995, 999, 1454, 1554, 1295, 983, 955, 1840, 1868, 1880,  
 1948, 1784, 1788, 586, 1800, 1804, 1864, 1768, 1141, 757, 773,  
 1482, 1486, 963, 478, 558, 565, 414, 423, 431, 233, 256,  
 1331, 1046, 1740, 1752, 1756, 911, 1912, 1339, 1418, 1038, 1042,  
 629, 672, 761, 336, 395, 199, 24, 32, 48, 81, 341,  
 1267, 1271, 1255, 1355, 1514, 1518, 1149, 1231, 1235, 1638, 1299,  
 1303, 1323, 1606, 1716, 1688, 1692, 1391, 1395, 1668, 1816, 1662,  
 1430, 1434, 1860, 991, 641, 523, 537, 346, 1684, 237, 245,  
 259, 296, 1442, 1185, 809, 749, 765, 1808, 266, 269, 1263,  
 1582, 1586, 841, 845, 1884, 1888, 1570, 1574, 1347, 1371, 1407,  
 1101, 1105, 1109, 1153, 1247, 1251, 404, 1426, 252, 1696, 1772,  
 1760, 1764, 1530, 1121, 919, 1219, 682, 439, 209, 117, 1522,  
 377, 1832, 666, 445, 227, 1646, 579, 292, 1630, 1916, 1169,  
 813, 817, 821, 959, 451, 220, 196, 1283, 1335, 971, 975,  
 1736, 1654, 785, 1610, 1666, 1446, 1387, 1026, 1085, 805, 915,  
 654, 1223, 987, 1920, 1708, 1550, 1093, 907, 180, 263, 241,  
 166, 1952, 1634, 1034, 895, 769, 311, 63, 1672, 863, 693,  
 272, 135, 192, 87, 923, 623, 697, 516, 1506, 1626, 1343,  
 1812, 709, 206, 1137, 1450, 1205, 1209, 903, 1275, 494, 105,  
 56, 1287, 1776, 1538, 1542, 1351, 1940, 883, 887, 781, 1383,  
 1030, 1081, 1117, 899, 927, 931, 642, 688, 1359, 753, 1367,  
 1642, 1848, 741, 15, 1896, 1053, 552, 387, 276, 93, 99,  
 1197, 1201, 1327, 701, 1700, 1748, 1239, 1243, 1744, 1458, 1546,  
 1712, 1872, 1466, 1113, 1307, 1311, 1097, 733, 737, 943, 1279,  
 5, 1069, 1073, 837, 248, 1422, 1363, 891, 576, 597, 1892,  
 1189, 230, 721, 713, 1590, 1594, 1438, 833, 853, 1562, 1566,  
 1856, 855, 157, 1049, 1502, 284, 288, 1089, 607, 308, 184,  
 1003, 1007, 1844, 871, 875, 1478, 1650, 849, 1836, 777, 1177,  
 1291, 1490, 825, 1614, 1618, 772, 1658, 867, 1399, 1403, 1526,  
 1022, 935, 1558, 1411, 457, 1161, 1165, 859, 1415, 224, 1720,  
 1510, 829, 1820, 789, 717, 202, 280, 162, 1227, 1057, 1796,  
 967, 793, 213, 615, 188, 1622, 1792, 1157, 1828, 1077, 951,  
 676, 1932, 1936, 939, 801, 153, 1211, 593, 1578, 1824, 879],  
 dtype=int64)

array([2016, 2015, 2014, 2013, 2012, 2011, 2009, 2010, 2017, 2008, 2007,  
 2006], dtype=int64)

array(['63%', '70%', '60%', '80%', '88%', '72%', '55%', '75%', '65%',  
 '85%', '73%', '64%', '66%', '68%', '50%', '100%', '77%', '90%',  
 '71%', '83%', '78%', '74%', '76%', '86%', '82%', '69%', '91%',  
 '42%', '61%', '73.5%', '62%', '67%', '58%', '60.5%', '79%', '81%',  
 '57%', '72.5%', '56%', '46%', '89%', '99%', '84%', '53%', '87%'],  
 dtype=object)

array(['France', 'U.S.A.', 'Fiji', 'Ecuador', 'Mexico', 'Switzerland',  
 'Netherlands', 'Spain', 'Peru', 'Canada', 'Italy', 'Brazil',  
 'U.K.', 'Australia', 'Wales', 'Belgium', 'Germany', 'Russia',  
 'Puerto Rico', 'Venezuela', 'Colombia', 'Japan', 'New Zealand',  
 'Costa Rica', 'South Korea', 'Amsterdam', 'Scotland', 'Martinique',  
 'Sao Tome', 'Argentina', 'Guatemala', 'South Africa', 'Bolivia',  
 'St. Lucia', 'Portugal', 'Singapore', 'Denmark', 'Vietnam',  
 'Grenada', 'Israel', 'India', 'Czech Republic',  
 'Domincan Republic', 'Finland', 'Madagascar', 'Philippines',  
 'Sweden', 'Poland', 'Austria', 'Honduras', 'Nicaragua',  
 'Lithuania', 'Niacragua', 'Chile', 'Ghana', 'Iceland', 'Eucador',  
 'Hungary', 'Suriname', 'Ireland'], dtype=object)

array([3.75, 2.75, 3. , 3.5 , 4. , 3.25, 2.5 , 5. , 1.75, 1.5 , 2.25,  
 2. , 1. ])

array(['\xa0', 'Criollo', 'Trinitario', 'Forastero (Arriba)', 'Forastero',  
 'Forastero (Nacional)', 'Criollo, Trinitario',  
 'Criollo (Porcelana)', 'Blend', 'Trinitario (85% Criollo)',  
 'Forastero (Catongo)', 'Forastero (Parazinho)',  
 'Trinitario, Criollo', 'CCN51', 'Criollo (Ocumare)', 'Nacional',  
 'Criollo (Ocumare 61)', 'Criollo (Ocumare 77)',  
 'Criollo (Ocumare 67)', 'Criollo (Wild)', 'Beniano', 'Amazon mix',  
 'Trinitario, Forastero', 'Forastero (Arriba) ASS', 'Criollo, +',  
 'Amazon', 'Amazon, ICS', 'EET', 'Blend-Forastero,Criollo',  
 'Trinitario (Scavina)', 'Criollo, Forastero', 'Matina',  
 'Forastero(Arriba, CCN)', 'Nacional (Arriba)',  
 'Forastero (Arriba) ASSS', 'Forastero, Trinitario',  
 'Forastero (Amelonado)', nan, 'Trinitario, Nacional',  
 'Trinitario (Amelonado)', 'Trinitario, TCGA', 'Criollo (Amarru)'],  
 dtype=object)

array(['Sao Tome', 'Togo', 'Peru', 'Venezuela', 'Cuba', 'Panama',  
 'Madagascar', 'Brazil', 'Ecuador', 'Colombia', 'Burma',  
 'Papua New Guinea', 'Bolivia', 'Fiji', 'Mexico', 'Indonesia',  
 'Trinidad', 'Vietnam', 'Nicaragua', 'Tanzania',  
 'Dominican Republic', 'Ghana', 'Belize', '\xa0', 'Jamaica',  
 'Grenada', 'Guatemala', 'Honduras', 'Costa Rica',  
 'Domincan Republic', 'Haiti', 'Congo', 'Philippines', 'Malaysia',  
 'Dominican Rep., Bali', 'Venez,Africa,Brasil,Peru,Mex', 'Gabon',  
 'Ivory Coast', 'Carribean', 'Sri Lanka', 'Puerto Rico', 'Uganda',  
 'Martinique', 'Sao Tome & Principe', 'Vanuatu', 'Australia',  
 'Liberia', 'Ecuador, Costa Rica', 'West Africa', 'Hawaii',  
 'St. Lucia', 'Cost Rica, Ven', 'Peru, Madagascar',  
 'Venezuela, Trinidad', 'Trinidad, Tobago',  
 'Ven, Trinidad, Ecuador', 'South America, Africa', 'India',  
 'Africa, Carribean, C. Am.', 'Tobago', 'Ven., Indonesia, Ecuad.',  
 'Trinidad-Tobago', 'Peru, Ecuador, Venezuela',  
 'Venezuela, Dom. Rep.', 'Colombia, Ecuador', 'Solomon Islands',  
 'Nigeria', 'Peru, Belize', 'Peru, Mad., Dom. Rep.', nan,  
 'PNG, Vanuatu, Mad', 'El Salvador', 'South America', 'Samoa',  
 'Ghana, Domin. Rep', 'Trinidad, Ecuador', 'Cameroon',  
 'Venezuela, Java', 'Venezuela/ Ghana', 'Venezuela, Ghana',  
 'Indonesia, Ghana', 'Peru(SMartin,Pangoa,nacional)', 'Principe',  
 'Central and S. America', 'Ven., Trinidad, Mad.',  
 'Carribean(DR/Jam/Tri)', 'Ghana & Madagascar',  
 'Ven.,Ecu.,Peru,Nic.', 'Madagascar & Ecuador',  
 'Guat., D.R., Peru, Mad., PNG', 'Peru, Dom. Rep',  
 'Dom. Rep., Madagascar', 'Gre., PNG, Haw., Haiti, Mad',  
 'Mad., Java, PNG', 'Ven, Bolivia, D.R.', 'DR, Ecuador, Peru',  
 'Suriname', 'Peru, Ecuador', 'Ecuador, Mad., PNG',  
 'Ghana, Panama, Ecuador', 'Venezuela, Carribean'], dtype=object)

```{python}  
df.info()  
```

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 1795 entries, 0 to 1794  
Data columns (total 9 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 Company 1795 non-null object   
 1 Specific Bean Origin or Bar Name 1795 non-null object   
 2 Reference Number 1795 non-null int64   
 3 Review Date 1795 non-null int64   
 4 Cocoa Percent 1795 non-null object   
 5 Company Location 1795 non-null object   
 6 Rating 1795 non-null float64  
 7 Bean Type 1794 non-null object   
 8 Broad Bean Origin 1794 non-null object   
dtypes: float64(1), int64(2), object(6)  
memory usage: 126.3+ KB

```{python}  
# Fill unknown and unformatted values with proper ones for readability and to  
# improve data accuracy and relevance   
df["Bean Type"].fillna("N/A", inplace=True)  
df["Bean Type"].replace("\xa0", "N/A", inplace=True)  
  
df["Broad Bean Origin"].fillna("N/A", inplace=True)  
df["Broad Bean Origin"].replace("\xa0", "N/A", inplace=True)  
  
df  
```

|  | Company | Specific Bean Origin or Bar Name | Reference Number | Review Date | Cocoa Percent | Company Location | Rating | Bean Type | Broad Bean Origin |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | A. Morin | Agua Grande | 1876 | 2016 | 63% | France | 3.75 | N/A | Sao Tome |
| 1 | A. Morin | Kpime | 1676 | 2015 | 70% | France | 2.75 | N/A | Togo |
| 2 | A. Morin | Atsane | 1676 | 2015 | 70% | France | 3.00 | N/A | Togo |
| 3 | A. Morin | Akata | 1680 | 2015 | 70% | France | 3.50 | N/A | Togo |
| 4 | A. Morin | Quilla | 1704 | 2015 | 70% | France | 3.50 | N/A | Peru |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 1790 | Zotter | Peru | 647 | 2011 | 70% | Austria | 3.75 | N/A | Peru |
| 1791 | Zotter | Congo | 749 | 2011 | 65% | Austria | 3.00 | Forastero | Congo |
| 1792 | Zotter | Kerala State | 749 | 2011 | 65% | Austria | 3.50 | Forastero | India |
| 1793 | Zotter | Kerala State | 781 | 2011 | 62% | Austria | 3.25 | N/A | India |
| 1794 | Zotter | Brazil, Mitzi Blue | 486 | 2010 | 65% | Austria | 3.00 | N/A | Brazil |

```{python}  
df.info()  
```

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 1795 entries, 0 to 1794  
Data columns (total 9 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 Company 1795 non-null object   
 1 Specific Bean Origin or Bar Name 1795 non-null object   
 2 Reference Number 1795 non-null int64   
 3 Review Date 1795 non-null int64   
 4 Cocoa Percent 1795 non-null object   
 5 Company Location 1795 non-null object   
 6 Rating 1795 non-null float64  
 7 Bean Type 1795 non-null object   
 8 Broad Bean Origin 1795 non-null object   
dtypes: float64(1), int64(2), object(6)  
memory usage: 126.3+ KB

```{python}  
# Check to make sure if all NaN and also any unpreferred / unformatted values  
# are resolved now  
print(df.isnull().sum())  
(df.isnull().sum() / df.shape[0]) \* 100  
```

Company 0  
Specific Bean Origin or Bar Name 0  
Reference Number 0  
Review Date 0  
Cocoa Percent 0  
Company Location 0  
Rating 0  
Bean Type 0  
Broad Bean Origin 0  
dtype: int64

Company 0.0  
Specific Bean Origin or Bar Name 0.0  
Reference Number 0.0  
Review Date 0.0  
Cocoa Percent 0.0  
Company Location 0.0  
Rating 0.0  
Bean Type 0.0  
Broad Bean Origin 0.0  
dtype: float64

```{python}  
# Rename "Cocoa Percent" column entries to be parsable by being numerical now  
df["Cocoa Percent"] = df["Cocoa Percent"].str.replace("%", "")  
df["Cocoa Percent"] = df["Cocoa Percent"].apply("float64")  
print(df["Cocoa Percent"].dtype)  
df["Cocoa Percent"].unique()  
```

float64

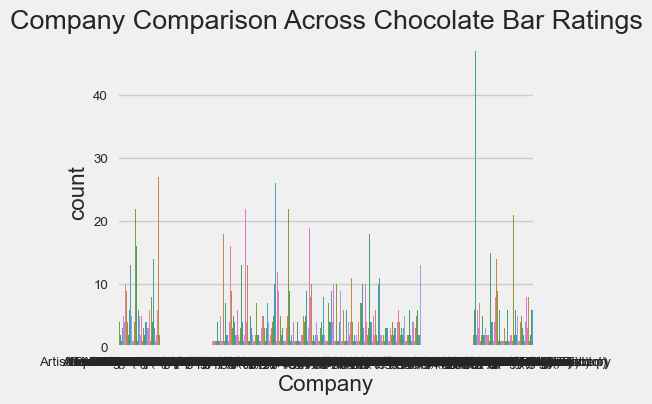
array([ 63. , 70. , 60. , 80. , 88. , 72. , 55. , 75. , 65. ,  
 85. , 73. , 64. , 66. , 68. , 50. , 100. , 77. , 90. ,  
 71. , 83. , 78. , 74. , 76. , 86. , 82. , 69. , 91. ,  
 42. , 61. , 73.5, 62. , 67. , 58. , 60.5, 79. , 81. ,  
 57. , 72.5, 56. , 46. , 89. , 99. , 84. , 53. , 87. ])

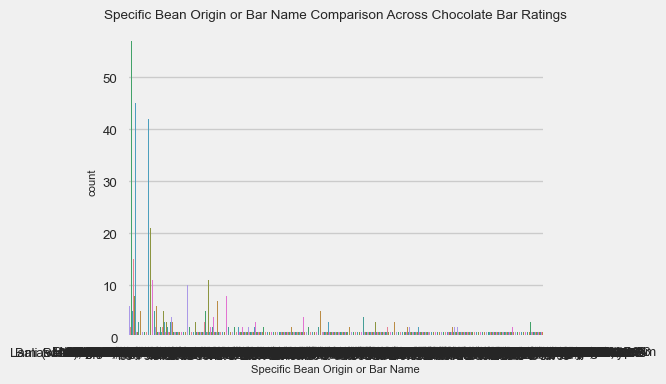
```{python}  
df.info()  
```

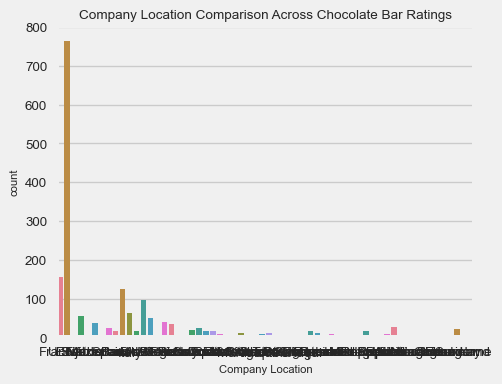
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 1795 entries, 0 to 1794  
Data columns (total 9 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 Company 1795 non-null object   
 1 Specific Bean Origin or Bar Name 1795 non-null object   
 2 Reference Number 1795 non-null int64   
 3 Review Date 1795 non-null int64   
 4 Cocoa Percent 1795 non-null float64  
 5 Company Location 1795 non-null object   
 6 Rating 1795 non-null float64  
 7 Bean Type 1795 non-null object   
 8 Broad Bean Origin 1795 non-null object   
dtypes: float64(2), int64(2), object(5)  
memory usage: 126.3+ KB

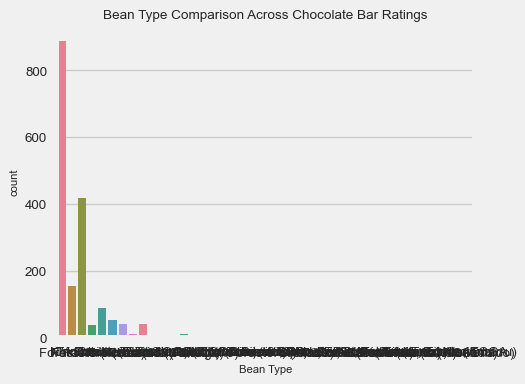
```{python}  
# Create bar graphs for descriptive statistics of the cars, figuring out how  
# many fall into which group within each qualitative cateogry  
def create\_bar\_graphs(attribute: str):  
 sns.countplot(x=attribute, data=df, palette=sns.color\_palette("husl", 8))  
 plt.title(f"{attribute} Comparison Across Chocolate Bar Ratings")  
 plt.rcParams["font.size"] = 7  
 plt.show()  
  
categorical\_columns = ["Company", "Specific Bean Origin or Bar Name", "Company Location", "Bean Type", "Broad Bean Origin", "Review Date"]  
  
for col in categorical\_columns:  
 create\_bar\_graphs(col)  
```

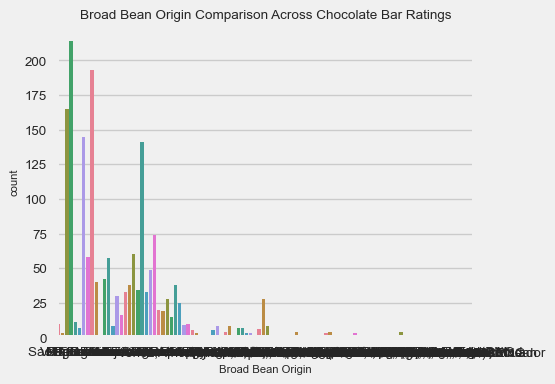
C:\Users\andre\AppData\Local\Temp\ipykernel\_25160\674198873.py:4: FutureWarning:   
  
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.  
  
 sns.countplot(x=attribute, data=df, palette=sns.color\_palette("husl", 8))  
C:\Users\andre\AppData\Local\Temp\ipykernel\_25160\674198873.py:4: UserWarning:   
The palette list has fewer values (8) than needed (416) and will cycle, which may produce an uninterpretable plot.  
 sns.countplot(x=attribute, data=df, palette=sns.color\_palette("husl", 8))  
C:\Users\andre\AppData\Local\Programs\Python\Python311\Lib\site-packages\IPython\core\pylabtools.py:152: UserWarning: Glyph 65533 (\N{REPLACEMENT CHARACTER}) missing from current font.  
 fig.canvas.print\_figure(bytes\_io, \*\*kw)  
C:\Users\andre\AppData\Local\Temp\ipykernel\_25160\674198873.py:4: FutureWarning:   
  
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.  
  
 sns.countplot(x=attribute, data=df, palette=sns.color\_palette("husl", 8))  
C:\Users\andre\AppData\Local\Temp\ipykernel\_25160\674198873.py:4: UserWarning:   
The palette list has fewer values (8) than needed (1039) and will cycle, which may produce an uninterpretable plot.  
 sns.countplot(x=attribute, data=df, palette=sns.color\_palette("husl", 8))  
C:\Users\andre\AppData\Local\Temp\ipykernel\_25160\674198873.py:4: FutureWarning:   
  
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.  
  
 sns.countplot(x=attribute, data=df, palette=sns.color\_palette("husl", 8))  
C:\Users\andre\AppData\Local\Temp\ipykernel\_25160\674198873.py:4: UserWarning:   
The palette list has fewer values (8) than needed (60) and will cycle, which may produce an uninterpretable plot.  
 sns.countplot(x=attribute, data=df, palette=sns.color\_palette("husl", 8))  
C:\Users\andre\AppData\Local\Temp\ipykernel\_25160\674198873.py:4: FutureWarning:   
  
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.  
  
 sns.countplot(x=attribute, data=df, palette=sns.color\_palette("husl", 8))  
C:\Users\andre\AppData\Local\Temp\ipykernel\_25160\674198873.py:4: UserWarning:   
The palette list has fewer values (8) than needed (41) and will cycle, which may produce an uninterpretable plot.  
 sns.countplot(x=attribute, data=df, palette=sns.color\_palette("husl", 8))  
C:\Users\andre\AppData\Local\Temp\ipykernel\_25160\674198873.py:4: FutureWarning:   
  
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.  
  
 sns.countplot(x=attribute, data=df, palette=sns.color\_palette("husl", 8))  
C:\Users\andre\AppData\Local\Temp\ipykernel\_25160\674198873.py:4: UserWarning:   
The palette list has fewer values (8) than needed (100) and will cycle, which may produce an uninterpretable plot.  
 sns.countplot(x=attribute, data=df, palette=sns.color\_palette("husl", 8))  
C:\Users\andre\AppData\Local\Temp\ipykernel\_25160\674198873.py:4: FutureWarning:   
  
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.  
  
 sns.countplot(x=attribute, data=df, palette=sns.color\_palette("husl", 8))  
C:\Users\andre\AppData\Local\Temp\ipykernel\_25160\674198873.py:4: UserWarning:   
The palette list has fewer values (8) than needed (12) and will cycle, which may produce an uninterpretable plot.  
 sns.countplot(x=attribute, data=df, palette=sns.color\_palette("husl", 8))

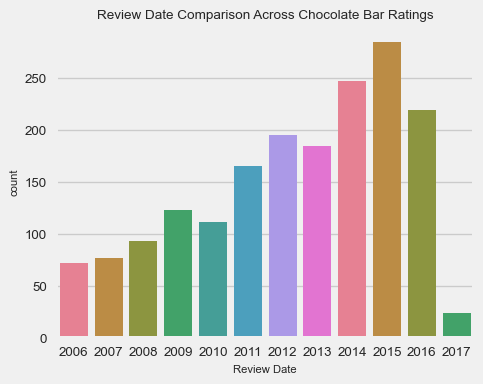










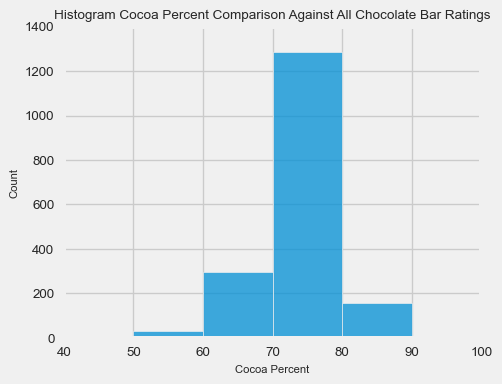


```{python}  
review\_date\_min\_year\_val = df["Review Date"].min()  
df["Review Date"] = df["Review Date"].map(lambda x: x - review\_date\_min\_year\_val)  
df["Review Date"].unique()  
```

array([10, 9, 8, 7, 6, 5, 3, 4, 11, 2, 1, 0], dtype=int64)

```{python}  
# Histogram plot illustrating the Cocoa Percent  
bin\_width = 10  
df\_cocoa\_percent = df["Cocoa Percent"].round(0)  
df\_cocoa\_percent = df\_cocoa\_percent.apply(int)  
hist\_low\_range = (min(df\_cocoa\_percent) // 10) \* 10  
hist\_high\_range = (max(df\_cocoa\_percent) // 10) \* 10  
  
# # Set the style of seaborn  
# sns.set(style="whitegrid")  
  
sns.histplot(df["Cocoa Percent"],   
 bins=range(hist\_low\_range, hist\_high\_range, bin\_width),  
 kde=False,  
 palette=sns.color\_palette("husl", 8))  
plt.rcParams["font.size"] = 7  
plt.title("Histogram Cocoa Percent Comparison Against All Chocolate Bar Ratings")  
plt.xlim(40, 100)  
plt.ylim(0, 1400)  
plt.show()  
```

C:\Users\andre\AppData\Local\Temp\ipykernel\_25160\3503189455.py:11: UserWarning: Ignoring `palette` because no `hue` variable has been assigned.  
 sns.histplot(df["Cocoa Percent"],



```{python}  
# Convert any needed categorical columns into numerical ones via factorizing (integer mapping)  
for col in df.columns:  
 if not pd.api.types.is\_numeric\_dtype(df[col]):  
 df[col] = pd.factorize(df[col])[0]  
```

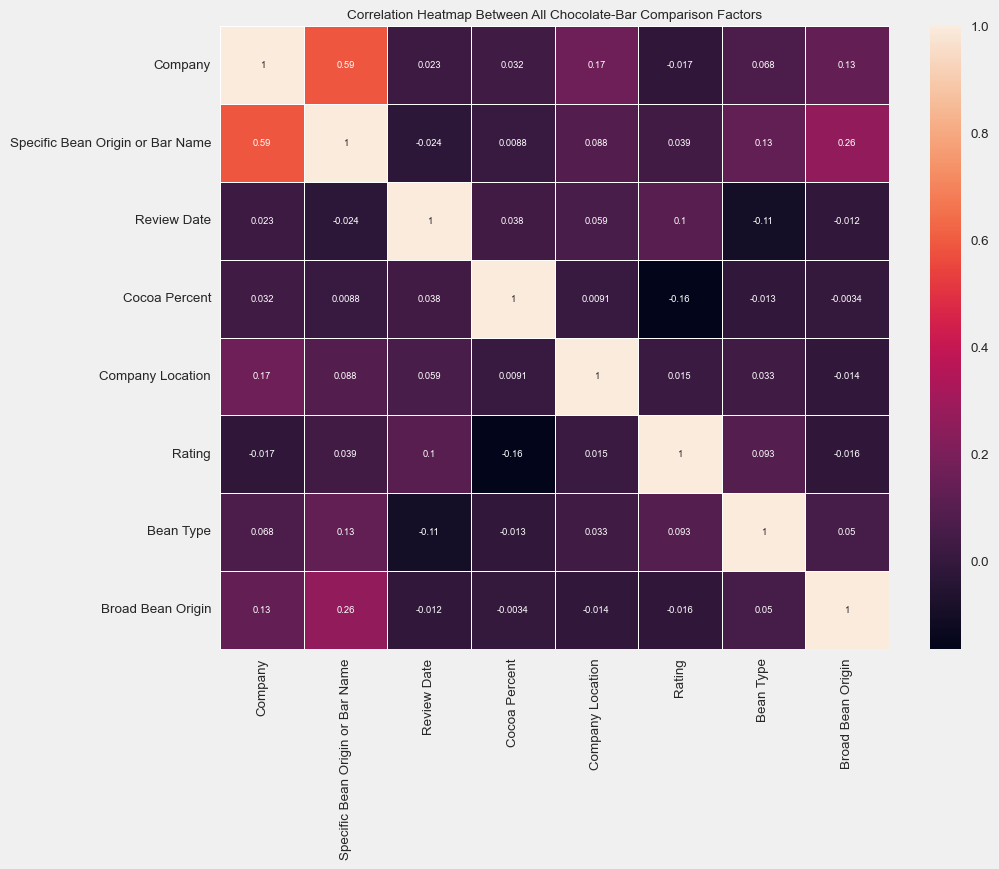
```{python}  
df  
```

|  | Company | Specific Bean Origin or Bar Name | Reference Number | Review Date | Cocoa Percent | Company Location | Rating | Bean Type | Broad Bean Origin |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 1876 | 10 | 63.0 | 0 | 3.75 | 0 | 0 |
| 1 | 0 | 1 | 1676 | 9 | 70.0 | 0 | 2.75 | 0 | 1 |
| 2 | 0 | 2 | 1676 | 9 | 70.0 | 0 | 3.00 | 0 | 1 |
| 3 | 0 | 3 | 1680 | 9 | 70.0 | 0 | 3.50 | 0 | 1 |
| 4 | 0 | 4 | 1704 | 9 | 70.0 | 0 | 3.50 | 0 | 2 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 1790 | 415 | 21 | 647 | 5 | 70.0 | 48 | 3.75 | 0 | 2 |
| 1791 | 415 | 100 | 749 | 5 | 65.0 | 48 | 3.00 | 4 | 31 |
| 1792 | 415 | 1037 | 749 | 5 | 65.0 | 48 | 3.50 | 4 | 57 |
| 1793 | 415 | 1037 | 781 | 5 | 62.0 | 48 | 3.25 | 0 | 57 |
| 1794 | 415 | 1038 | 486 | 4 | 65.0 | 48 | 3.00 | 0 | 7 |

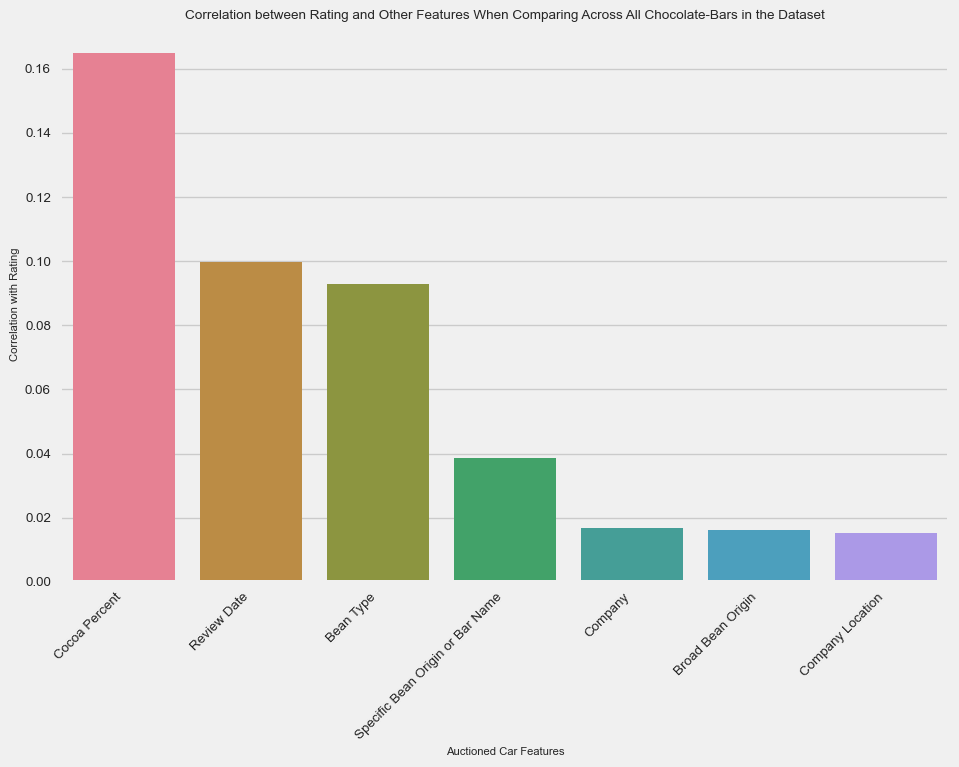
```{python}  
# Removed unnecessary columns not needed for heatmap comparison  
df = df.drop(["Reference Number"], axis=1)  
df  
```

|  | Company | Specific Bean Origin or Bar Name | Review Date | Cocoa Percent | Company Location | Rating | Bean Type | Broad Bean Origin |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 10 | 63.0 | 0 | 3.75 | 0 | 0 |
| 1 | 0 | 1 | 9 | 70.0 | 0 | 2.75 | 0 | 1 |
| 2 | 0 | 2 | 9 | 70.0 | 0 | 3.00 | 0 | 1 |
| 3 | 0 | 3 | 9 | 70.0 | 0 | 3.50 | 0 | 1 |
| 4 | 0 | 4 | 9 | 70.0 | 0 | 3.50 | 0 | 2 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 1790 | 415 | 21 | 5 | 70.0 | 48 | 3.75 | 0 | 2 |
| 1791 | 415 | 100 | 5 | 65.0 | 48 | 3.00 | 4 | 31 |
| 1792 | 415 | 1037 | 5 | 65.0 | 48 | 3.50 | 4 | 57 |
| 1793 | 415 | 1037 | 5 | 62.0 | 48 | 3.25 | 0 | 57 |
| 1794 | 415 | 1038 | 4 | 65.0 | 48 | 3.00 | 0 | 7 |

```{python}  
# Correlation heatmap to quantify relationships between chocolate-bar comparison  
# attributes  
plt.figure(figsize=(10, 8))  
plt.rcParams["font.size"] = 7  
sns.heatmap(df.corr(), annot=True, linewidths=0.5)  
plt.title("Correlation Heatmap Between All Chocolate-Bar Comparison Factors")  
plt.show()  
  
# Correlation bar graph between Rating and all other chocolate-bar comparison  
# attributes  
target\_corr = df.corr()["Rating"].abs().sort\_values(ascending=False)  
plt.figure(figsize=(10, 8))  
plt.rcParams["font.size"] = 7  
sns.barplot(x=target\_corr.index[1:], y=target\_corr.values[1:], palette=sns.color\_palette("husl", 8))  
plt.xticks(rotation=45, ha="right")  
plt.xlabel("Auctioned Car Features")  
plt.ylabel("Correlation with Rating")  
plt.title("Correlation between Rating and Other Features When Comparing Across All Chocolate-Bars in the Dataset")  
plt.tight\_layout()  
plt.show()  
```



C:\Users\andre\AppData\Local\Temp\ipykernel\_25160\3963744686.py:14: FutureWarning:   
  
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.  
  
 sns.barplot(x=target\_corr.index[1:], y=target\_corr.values[1:], palette=sns.color\_palette("husl", 8))  
C:\Users\andre\AppData\Local\Temp\ipykernel\_25160\3963744686.py:14: UserWarning: The palette list has more values (8) than needed (7), which may not be intended.  
 sns.barplot(x=target\_corr.index[1:], y=target\_corr.values[1:], palette=sns.color\_palette("husl", 8))



```{python}  
# Removed unnecessary columns not needed for Machine Learning analysis  
df = df.drop(columns=["Company", "Broad Bean Origin", "Company Location", "Specific Bean Origin or Bar Name"], axis=1)  
df  
```

|  | Review Date | Cocoa Percent | Rating | Bean Type |
| --- | --- | --- | --- | --- |
| 0 | 10 | 63.0 | 3.75 | 0 |
| 1 | 9 | 70.0 | 2.75 | 0 |
| 2 | 9 | 70.0 | 3.00 | 0 |
| 3 | 9 | 70.0 | 3.50 | 0 |
| 4 | 9 | 70.0 | 3.50 | 0 |
| ... | ... | ... | ... | ... |
| 1790 | 5 | 70.0 | 3.75 | 0 |
| 1791 | 5 | 65.0 | 3.00 | 4 |
| 1792 | 5 | 65.0 | 3.50 | 4 |
| 1793 | 5 | 62.0 | 3.25 | 0 |
| 1794 | 4 | 65.0 | 3.00 | 0 |

## Machine Learning - Model Training and Evaluation

Great, now we are onto the Machine Learning part of the blog post!

Since the dataframe is now properly cleaned, sorted, and integer-mapped by this point, I had split the respective dataframe into the train and test datasets for the Machine Learning model with 80% going to the training dataset and the last 20% going to the test dataset. Fortunately, because order of the data sequentially does not matter here, I was able to utilize the train\_test\_split function for shuffling and randomization, making the future-generated Machine Learning model more unpredictable but also more objective in its returned model results.

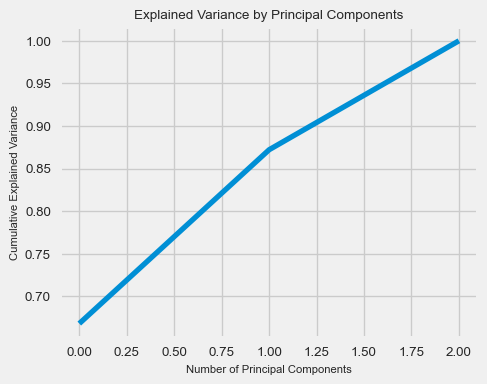
Note that here I also used a Pipeline object from the scikit-learn package as well as the StandardScaler classes. On one hand, the MinMaxScaler class is useful for scaling columns to a specific range, usually between [0, 1], to maintain consistency. On the other hand, the StandardScaler class is useful for apply Z-score normalization / transformation on the data to avoid sensivite-prone Machine Learning algorithms which require appropriate scaling of the features within its trained dataset. As I learned from online, this Pipeline object is necesary to ensure appropriate preprocessing just before the dataset is passed to the Machine Learning model for training and later evaluation.

```{python}  
X = df.drop(columns=["Rating"], axis=1)  
y = df["Rating"]  
  
print("X Shape:", X.shape)  
print("Y Shape:", y.shape)  
  
pipeline = Pipeline([  
 ("std\_scaler", StandardScaler()),  
 ("min\_max\_scaler", MinMaxScaler())  
])  
  
X\_scaled = pipeline.fit\_transform(X)  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, shuffle=True, random\_state=42)  
```

X Shape: (1795, 3)  
Y Shape: (1795,)

Due to the higher-order dimensionality in the X columns, I had to utilze PCA to reduce the X columns in the dataset down to a manageable amount for humans (most accepted is 2 principal components).

```{python}  
# Used PCA to lower the dimensionality of the dataset used later for Machine  
# Learning (training and testing)  
pca = PCA()  
principal\_components = pca.fit\_transform(X\_test)  
plt.plot(np.cumsum(pca.explained\_variance\_ratio\_))  
plt.title("Explained Variance by Principal Components")  
plt.xlabel("Number of Principal Components")  
plt.ylabel("Cumulative Explained Variance")  
plt.show()  
```



```{python}  
# Created a new dataframe containing the 2 main principal components used in the  
# later Machine Learning section of this blog-post  
pca\_df = pd.DataFrame(data=principal\_components[:, :2], columns=["PC1", "PC2"])  
pca\_df  
```

|  | PC1 | PC2 |
| --- | --- | --- |
| 0 | 0.303283 | -0.053220 |
| 1 | 0.032318 | -0.015645 |
| 2 | -0.152118 | -0.054858 |
| 3 | -0.242767 | -0.047512 |
| 4 | -0.152118 | -0.054858 |
| ... | ... | ... |
| 354 | 0.573557 | -0.104226 |
| 355 | -0.333623 | -0.044195 |
| 356 | -0.240286 | -0.006366 |
| 357 | -0.234078 | 0.100101 |
| 358 | -0.242870 | -0.049526 |

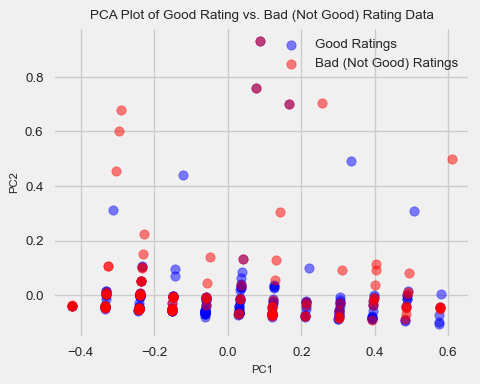
```{python}  
y\_test.value\_counts()  
```

Rating  
3.50 76  
3.25 70  
3.00 69  
2.75 50  
3.75 38  
2.50 24  
4.00 16  
2.00 6  
2.25 5  
1.50 3  
1.00 1  
5.00 1  
Name: count, dtype: int64

Adding in the y column ("Rating") to this modified dataframe containing two new PCA components, we are able to experimentally look at some basic labeling of the data prior to clustering.

*(Disclaimer: I realize that clustering is considered unsupervised Machine Learning. The experimental practices of labels here are merely for reference to anticipate future clustering patterns. In real clustering applications, I realize that I wouldn’t really have the labels to cluster in the first place. As seen throughout this section, this practice doesn’t intend to skew my Machine Learning methods.)*

```{python}  
# Add the "Rating" column to the PCA DataFrame  
pca\_df["Rating"] = y\_test.values  
  
# Separate the data based on the "Rating" columns (my subjective cut-off on   
# numerical ratings to label "good" vs. "bad" ratings - out of 5)  
good\_ratings = pca\_df[pca\_df["Rating"] >= 3.5]  
not\_good\_ratings = pca\_df[pca\_df["Rating"] < 3.5]  
  
# Scatter plot with different colors for "Good" and "Bad"  
plt.scatter(not\_good\_ratings["PC1"], not\_good\_ratings["PC2"], c='blue', label='Good Ratings', alpha=0.5)  
plt.scatter(good\_ratings["PC1"], good\_ratings["PC2"], c='red', label='Bad (Not Good) Ratings', alpha=0.5)  
  
plt.title("PCA Plot of Good Rating vs. Bad (Not Good) Rating Data")  
plt.xlabel("PC1")  
plt.ylabel("PC2")  
plt.legend()  
plt.show()  
```



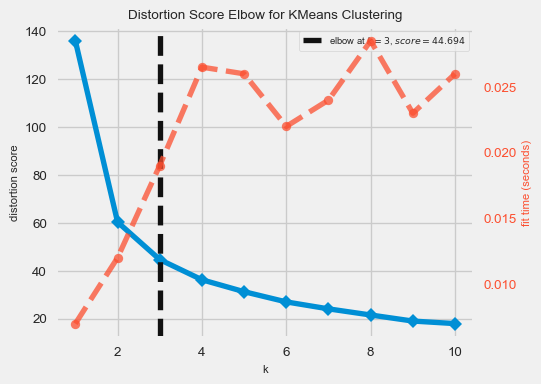
```{python}  
# Match PCA dataframe with y\_test data with "binary" classification of chocolate-bar  
# ratings  
y\_test = y\_test.map(lambda entry: 0 if entry < 3.5 else 1)  
y\_test  
```

1234 0  
220 1  
1516 1  
438 1  
1268 0  
 ..  
1114 0  
1729 0  
1615 0  
1036 1  
964 0  
Name: Rating, Length: 359, dtype: int64

Now, we will do an exploratory K-Means Clustering to see if we detect any patterns or correlations in the dataset between Rating and other categories that were reported with the existing chocolate-bar evaluations.

```{python}  
# Using the KElbowVisualizer to determine the appropriate number of clusters  
# to use in the K-Means algorithm  
kmeans\_model = KMeans()  
kmeans\_elbow\_visualizer = KElbowVisualizer(kmeans\_model, k=(1, 11))  
kmeans\_elbow\_visualizer.fit(X\_train)  
kmeans\_elbow\_visualizer.show()  
```

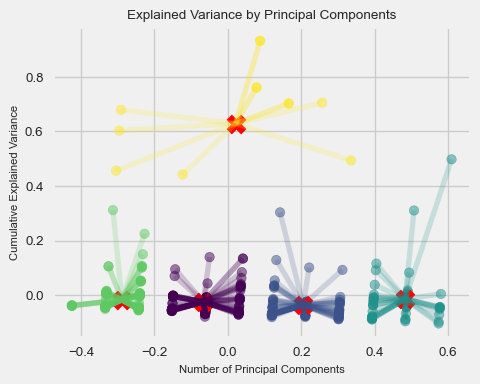
C:\Users\andre\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\\_kmeans.py:1416: FutureWarning: The default value of `n\_init` will change from 10 to 'auto' in 1.4. Set the value of `n\_init` explicitly to suppress the warning  
 super().\_check\_params\_vs\_input(X, default\_n\_init=10)  
C:\Users\andre\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\\_kmeans.py:1416: FutureWarning: The default value of `n\_init` will change from 10 to 'auto' in 1.4. Set the value of `n\_init` explicitly to suppress the warning  
 super().\_check\_params\_vs\_input(X, default\_n\_init=10)  
C:\Users\andre\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\\_kmeans.py:1416: FutureWarning: The default value of `n\_init` will change from 10 to 'auto' in 1.4. Set the value of `n\_init` explicitly to suppress the warning  
 super().\_check\_params\_vs\_input(X, default\_n\_init=10)  
C:\Users\andre\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\\_kmeans.py:1416: FutureWarning: The default value of `n\_init` will change from 10 to 'auto' in 1.4. Set the value of `n\_init` explicitly to suppress the warning  
 super().\_check\_params\_vs\_input(X, default\_n\_init=10)  
C:\Users\andre\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\\_kmeans.py:1416: FutureWarning: The default value of `n\_init` will change from 10 to 'auto' in 1.4. Set the value of `n\_init` explicitly to suppress the warning  
 super().\_check\_params\_vs\_input(X, default\_n\_init=10)  
C:\Users\andre\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\\_kmeans.py:1416: FutureWarning: The default value of `n\_init` will change from 10 to 'auto' in 1.4. Set the value of `n\_init` explicitly to suppress the warning  
 super().\_check\_params\_vs\_input(X, default\_n\_init=10)  
C:\Users\andre\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\\_kmeans.py:1416: FutureWarning: The default value of `n\_init` will change from 10 to 'auto' in 1.4. Set the value of `n\_init` explicitly to suppress the warning  
 super().\_check\_params\_vs\_input(X, default\_n\_init=10)  
C:\Users\andre\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\\_kmeans.py:1416: FutureWarning: The default value of `n\_init` will change from 10 to 'auto' in 1.4. Set the value of `n\_init` explicitly to suppress the warning  
 super().\_check\_params\_vs\_input(X, default\_n\_init=10)  
C:\Users\andre\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\\_kmeans.py:1416: FutureWarning: The default value of `n\_init` will change from 10 to 'auto' in 1.4. Set the value of `n\_init` explicitly to suppress the warning  
 super().\_check\_params\_vs\_input(X, default\_n\_init=10)  
C:\Users\andre\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\\_kmeans.py:1416: FutureWarning: The default value of `n\_init` will change from 10 to 'auto' in 1.4. Set the value of `n\_init` explicitly to suppress the warning  
 super().\_check\_params\_vs\_input(X, default\_n\_init=10)



<Axes: title={'center': 'Distortion Score Elbow for KMeans Clustering'}, xlabel='k', ylabel='distortion score'>

```{python}  
# Complete K-Means Clustering to find any clustering relationships in dataset  
# Run the kmeans model on scaled data  
kmeans\_model = KMeans(n\_clusters=5, random\_state=42).fit(X\_train)  
  
# Get the cluster number for each datapoint  
X\_clusters = kmeans\_model.predict(X\_test)  
  
# Save the cluster centroids  
X\_clusters\_centers = kmeans\_model.cluster\_centers\_  
  
# Plot PCA and K-Means cluster centers  
X\_test\_principal\_clusters\_centers = pca.transform(X\_clusters\_centers)  
plt.scatter(principal\_components[:, 0], principal\_components[:, 1], c=X\_clusters, cmap='viridis', s=50, alpha=0.5)  
plt.scatter(X\_test\_principal\_clusters\_centers[:, 0], X\_test\_principal\_clusters\_centers[:, 1], c='red', marker='X', s=200, label='Cluster Centers')  
  
# Set up a LineCollection (for creating lines from each data point to their   
# respective cluster center)  
lines = [[(x1, y1), (x2, y2)] for x1, y1, x2, y2 in zip(principal\_components[:, 0], principal\_components[:, 1], X\_test\_principal\_clusters\_centers[X\_clusters][:, 0], X\_test\_principal\_clusters\_centers[X\_clusters][:, 1])]  
lc = LineCollection(lines, cmap='viridis', norm=plt.Normalize(vmin=np.min(X\_clusters), vmax=np.max(X\_clusters)), alpha=0.2)  
  
# Set the color values based on the distance from the cluster center  
lc.set\_array(np.array(X\_clusters))  
  
# Add the created LineCollection to the plot  
plt.gca().add\_collection(lc)  
  
plt.title("Explained Variance by Principal Components")  
plt.xlabel("Number of Principal Components")  
plt.ylabel("Cumulative Explained Variance")  
plt.show()  
```

C:\Users\andre\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\\_kmeans.py:1416: FutureWarning: The default value of `n\_init` will change from 10 to 'auto' in 1.4. Set the value of `n\_init` explicitly to suppress the warning  
 super().\_check\_params\_vs\_input(X, default\_n\_init=10)

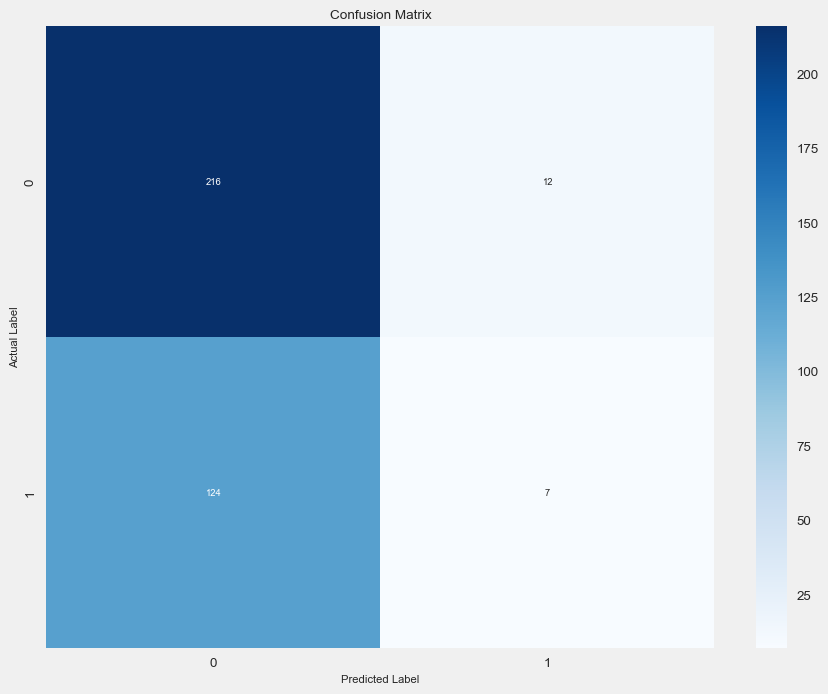


```{python}  
# Obtain predictions and calculate distance from cluster centroid (using Euclidean Distance)  
kmeans\_dist = [np.linalg.norm(x - y) for x, y in zip(principal\_components, X\_test\_principal\_clusters\_centers[X\_clusters])]  
  
y\_pred = np.array(kmeans\_dist)  
y\_pred[kmeans\_dist >= np.percentile(kmeans\_dist, 95)] = 1  
y\_pred[kmeans\_dist < np.percentile(kmeans\_dist, 95)] = 0  
y\_pred  
```

array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,  
 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,  
 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,  
 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,  
 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,  
 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.,  
 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.,  
 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,  
 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,  
 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 1., 0., 0.,  
 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.,  
 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,  
 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,  
 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,  
 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.,  
 0., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,  
 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,  
 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.,  
 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.,  
 0., 0., 0., 0., 0., 1., 0., 1., 0., 0., 0., 0., 0., 1., 1., 0., 0.,  
 0., 1., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.,  
 0., 0.])

```{python}  
# Display the accuracy statistics and the confusion matrix of the K-Means algorithm   
# cluster predictions  
clf\_report = pd.DataFrame(classification\_report(y\_true=y\_test, y\_pred=y\_pred, output\_dict=True, zero\_division=0))  
conf\_matrix = confusion\_matrix(y\_true=y\_test, y\_pred=y\_pred)  
  
print(f"ROC AUC Score: {roc\_auc\_score(y\_true=y\_test, y\_score=y\_pred) \* 100:.2f}%")  
print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")  
print(f"CLASSIFICATION REPORT:\n{clf\_report}")  
print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")  
print(f"Confusion Matrix:\n{conf\_matrix}")  
  
# Plot the confusion matrix  
plt.figure(figsize=(10, 8))  
plt.rcParams["font.size"] = 7  
sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues')  
plt.xlabel('Predicted Label')  
plt.ylabel('Actual Label')  
plt.title('Confusion Matrix')  
plt.show()  
```

ROC AUC Score: 50.04%  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
CLASSIFICATION REPORT:  
 0 1 accuracy macro avg weighted avg  
precision 0.635294 0.368421 0.62117 0.501858 0.537911  
recall 0.947368 0.053435 0.62117 0.500402 0.621170  
f1-score 0.760563 0.093333 0.62117 0.426948 0.517089  
support 228.000000 131.000000 0.62117 359.000000 359.000000  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Confusion Matrix:  
[[216 12]  
 [124 7]]



Finally, in addition to the K-Means clustering, for each entry in the (PCA) dataset, I have computed the distance to its closest cluster center and provided its respective closest cluster-index in a provided dataframe below, leaving this as an exercise to the reader to further explore any potential relationships that may be present here in the evalation scores.

```{python}  
# Calculates distances from mean and group similar-distances (print table)  
df\_kmeans\_cluster\_info = pd.DataFrame(data={"PC1": principal\_components[:, 0], 'PC2': principal\_components[:, 1]})  
  
# Find the index of the closest cluster center for each point  
closest\_cluster\_index = [np.argmin(np.linalg.norm(principal\_components[i] - X\_clusters\_centers, axis=1)) for i in range(len(principal\_components))]  
  
# Add the closest cluster center number and Euclidean distance to the DataFrame  
df\_kmeans\_cluster\_info["Distance\_to\_Closest\_Cluster"] = [np.linalg.norm(principal\_components[i] - X\_clusters\_centers[X\_clusters[i]]) for i in range(len(principal\_components))]  
df\_kmeans\_cluster\_info["Closest\_Cluster"] = X\_clusters[closest\_cluster\_index]  
df\_kmeans\_cluster\_info  
```

|  | PC1 | PC2 | Distance\_to\_Closest\_Cluster | Closest\_Cluster |
| --- | --- | --- | --- | --- |
| 0 | 0.303283 | -0.053220 | 0.558196 | 0 |
| 1 | 0.032318 | -0.015645 | 0.814719 | 0 |
| 2 | -0.152118 | -0.054858 | 0.983476 | 0 |
| 3 | -0.242767 | -0.047512 | 1.244446 | 0 |
| 4 | -0.152118 | -0.054858 | 0.983476 | 0 |
| ... | ... | ... | ... | ... |
| 354 | 0.573557 | -0.104226 | 0.794724 | 0 |
| 355 | -0.333623 | -0.044195 | 1.323382 | 0 |
| 356 | -0.240286 | -0.006366 | 1.234437 | 0 |
| 357 | -0.234078 | 0.100101 | 1.175465 | 0 |
| 358 | -0.242870 | -0.049526 | 1.244164 | 0 |

## Conclusions

* Given that (K-Means) Clustering is considered as Unsupervised Machine Learning, I found that the 3 kept columns from the original dataset - Cocoa Percent, Review Date, and Bean Type - were excellent in providing insights about the factors that went into the final determination of the Rating of the overall set of chocolate-bar entries. Looking at the dataset more closely, chocolate bars that had a similar numerical percentages of chocolate in its composition, similar dates of review, and similar-tasting chocolate bean types seemingly were grouped in the same cluster. This is something I expected going into this blog-post because this seems reasonable. However, from basic observation, no one factor dominated the other as I could find any leading trends in the K-Means clustering completed above.
* In terms of expanding the scope of this blog-post, I would have taken this further by viewing chocolate-bar entries that are similar in the distance from the same cluster center must be “numerically” similar and perform further Machine Learning analyses into determining the factors that lead to them receiving that particular score among the consensus of chocolate-bar evaluators.
* Ultimately, I learned a great deal from the blog post experience as I now better understand how to properly utilize K-Means Classifiers, an unsupervised classification method of Machine Learning, to find relationships between groups of data and infer conclusions about any potential trends, through applying it to a practical, every-day dilemma in our society. Even though the datasets in the real world are never this simple to analyze, I hope to utilize in future contexts as this was an interesting study to complete and creative question to answer. This does seem like a sub-field and problem space that I may try to pursue in the future.

## Reference Sources and Citations (IEEE Format)

To complete this blog post, I used the following online sources as references for developing this:

[1] Chocolate Car Ratings Dataset:

* R. Tatman, “Chocolate Bar Ratings”, 2017. [Online]. Available: https://www.kaggle.com/datasets/rtatman/chocolate-bar-ratings/data. [Accessed 15-Nov.-2023].

[2] KMeans Elbow Reference:

* F. Javier Gallego, “Outliers+EDA+Clustering Tutorial”, Jul.-2022. [Online]. Available: https://www.kaggle.com/code/javigallego/outliers-eda-clustering-tutorial. [Accessed 16-Nov.-2023].

[3] KMeans Reference:

* M. Isbaine, “Fraud Detection”, 2021. [Online]. Available: https://www.kaggle.com/code/mohamedisbaine/fraud-detection. [Accessed 16-Nov.-2023].